Topic 12:
Basic Ray Tracing

- Introduction to ray tracing
- Computing rays
- Computing intersections
  - ray-triangle
  - ray-polygon
  - ray-quadric
  - the scene signature
- Computing normals
- Evaluating shading model
- Spawning rays
- Incorporating transmission
  - refraction
  - ray-spawning & refraction

Computing Ray-Object Intersections

Basic loop:
for each pixel \( \overline{q} \)
1. cast ray \( r \) through \( \overline{q} \)
2. find 1st intersection of \( \overline{q} \) with scene (i.e., point \( \overline{p} \))
3. estimate amount of light reaching \( \overline{p} \)
4. estimate amount of light travelling from \( \overline{p} \) to \( \overline{q} \) along ray \( r \)
Computing Ray-Triangle Intersections

Algorithm #1

1. Compute normal
   \[ \overrightarrow{n} = (\overrightarrow{P_2 - P_1}) \times (\overrightarrow{P_3 - P_1}) \]

2. Compute intersection of ray and plane of triangle
   i.e., find \( \lambda \) that satisfies
   \[ [\overrightarrow{P_1 - P(\lambda)}] \cdot \overrightarrow{n} = 0 \]

3. Verify that \( P(\lambda) \) falls within triangle e.g., using half-space constraints (assignment 4)

Algorithm #2

1. Parameterize the triangle plane
   \[ P(\alpha, \beta) = P_1 + \alpha(\overrightarrow{P_2 - P_1}) + \beta(\overrightarrow{P_3 - P_1}) \]

2. Find \( \alpha, \beta, \lambda^* \) that satisfy
   \[ P(\alpha, \beta) = P(\lambda^*) \]

   \[ \Rightarrow \text{solve} \quad \begin{bmatrix} -(\overrightarrow{P_2 - P_1}) & -(\overrightarrow{P_3 - P_1}) & (\overrightarrow{q_w - P_1}) \end{bmatrix} \begin{bmatrix} \alpha \\ \beta \\ \lambda^* \end{bmatrix} = \overrightarrow{P_1 - P} \]

3. \( P(\lambda^*) \) inside triangle \( \iff \alpha > 0, \beta > 0, \alpha + \beta < 1 \)
Computing Ray-Polygon Intersections

1. Compute \( \vec{p}(x) \) using Algorithm 1 or 2 (by picking 3 adjacent non-collinear vertices)

2. Verify that \( p(x) \) lies inside the polygon
   - Convert the 3D polygon & \( p(x) \) to 2D
   - Do the verification in 2D (recall assignment #1)

Computing Ray-Poly Intersections: Step a

- Suppose \( \vec{u} \) is not along \( z \)-axis
- Project all vertices and \( p'(x) \) onto \( xy \)-plane:
  \[
  \vec{p}_i = \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} \rightarrow \begin{bmatrix} x_i' \\ y_i' \end{bmatrix} = \vec{p}'_i
  \]

\( \text{if } \vec{u} \text{ is along } z \text{-axis, project onto } xz \text{-plane} \)
Computing Ray-Poly Intersections: Step b

Key theorem:

If \( \vec{P} \) inside, every 2D half-line starting at \( \vec{P} \) must intersect the polygon's boundary an odd \# of times.

Verification algorithm:
1. Pick any non-vertex point on boundary.
2. Define lines \( \ell \) through \( \vec{P}, \vec{Q} \) and \( \ell_i \) through \( \vec{P}_i, \vec{P}_{i+1} \).
3. Intersect \( \ell \) with each \( \ell_i \).
4. Count intersections that are on the same side of \( \vec{P}_i \) and on the polygon boundary.

\[
\bar{Q} = \frac{1}{3}(\vec{P}_i + \vec{P}_{i+1})
\]
Computing Ray-Quadric Intersections

General implicit equation

\[
\begin{bmatrix} x & y & z & 1 \end{bmatrix} \begin{bmatrix} A & D & E & G \\ D & B & F & H \\ E & F & C & I \\ G & H & I & J \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = 0
\]

defined up to a scale factor

Computing Ray-Quadric Intersections

\[ \overrightarrow{r} = \overrightarrow{c} + \lambda(\overrightarrow{P_0} - \overrightarrow{c}) \]

Must solve the equation

\[
\begin{bmatrix} A & D & E & G \\ D & B & F & H \\ E & F & C & I \\ G & H & I & J \end{bmatrix} \begin{bmatrix} P(3) \\ P(4) \\ P(5) \\ P(6) \end{bmatrix} = 0
\]

expressed in homogeneous 3D coordinates

for a given quadric, this matrix is known
Computing Ray-Quadric Intersections: 3 Cases

\[ p(\lambda) = \overline{c} + \lambda(\overline{p}_w - \overline{c}) \]

\[ \Delta > 0 \Rightarrow 2 \text{ hits} \]
\[ \Delta < 0 \Rightarrow 0 \text{ hits} \]
\[ \Delta = 0 \Rightarrow 1 \text{ hit} \]

After expanding, we have a quadratic equation in terms of \( \alpha \):

\[ \alpha \alpha^2 + \beta \alpha + \gamma = 0 \]

Solution is \( \alpha = \frac{-\beta \pm \sqrt{\Delta}}{2\alpha} \), \( \Delta = \beta^2 - 4\alpha\gamma \)

Ray-Quadric Intersections: Sub-cases for \( \Delta > 0 \)

\( a, b < 0 \)

\( \Rightarrow \) hits are behind the viewplane

\( a > 0, a_2 > 0 \)

\( \Rightarrow \) 2 valid hits, smallest \( \beta \) gives intersection closest to camera

\( a_1, a_2 < 0 \)

\( \Rightarrow p(\alpha) \) is a valid hit

After expanding, we have a quadratic equation in terms of \( \alpha \):

\[ \alpha \alpha^2 + \beta \alpha + \gamma = 0 \]

Solution is \( \alpha = \frac{-\beta \pm \sqrt{\Delta}}{2\alpha} \), \( \Delta = \beta^2 - 4\alpha\gamma \)
Intersecting Rays & Composite Objects

\[ \mathbf{P}(\lambda) = \mathbf{c} + \lambda (\mathbf{P}_0 - \mathbf{c}) \]

- When an object is bounded by multiple parametric surfaces we must test for intersection with each of the components.
- Example: Cylinder = "Quadratic wall" + 2 planar caps.
  Cone = "Quadratic wall" + 1 planar base

⇒ See Leonid Sigal's slides for more details

---

Ray Intersection: Efficiency Considerations

Intersection tests can be very expensive!

⇒ Use data structures to avoid testing intersection with objects that clearly do not intersect.
Ray Intersection: Efficiency Considerations

Intersection tests can be very expensive!

⇒ Use data structures to avoid testing intersection with objects that clearly do not intersect

Examples:
- Test intersection with objects bounding volume first, test ray-object intersection only if ray intersects volume
- Apply this idea hierarchically, for part-based objects

Ray Intersection: Efficiency Considerations

Intersection tests can be very expensive!

⇒ Use data structures to avoid testing intersection with objects that clearly do not intersect

Examples:
- Image-space intersections: instead of intersecting ray & bounding volume, project volume & check whether pixel falls inside that projection
Topic 12:

Basic Ray Tracing

- Introduction to ray tracing
- Computing rays
- Computing intersections
  - ray-triangle
  - ray-polygon
  - ray-quadric
- the scene signature
- Computing normals
- Evaluating shading model
- Spawning rays
- Incorporating transmission
  - refraction
  - ray-spawning & refraction

The Scene Signature

Definition:

An image $S$ where $S(i,j)=k$ if object $k$ is the first object along ray through $(i,j)$

* The scene signature is a great tool for debugging and for testing intersection methods
The Scene Signature

Definition:
An image $S$ where $S(i,j) = k$ if object $k$ is the first object along ray through $(i,j)$

* The scene signature is a great tool for debugging and for testing intersection methods

Computing the Scene Signature

Definition:
An image $S$ where $S(i,j) = k$ if object $k$ is the first object along ray through $(i,j)$

Algorithm pseudocode:

for $i = 0$ to $Nrows - 1$
  for $j = 0$ to $Ncols - 1$
    construct ray through pixel $(i,j)$
    $a_{ij} = \infty$
    for $k = 0$ to $Nobjects$
      $a^* = \text{closest intersection of ray with object } k$
      if $a^* > 0$ and $a^* < a_{ij}$, set $a_{ij} = a^*$, $S(i,j) = k$
Computational Issues in Basic Ray Tracing

Basic loop:
for each pixel \( \vec{q} \)
1. cast ray \( r \) through \( \vec{q} \)
2. find 1st intersection of \( r \) with scene (i.e. point \( \vec{p} \))
3. estimate amount of light reaching \( \vec{p} \)

4. estimate amount of light travelling from \( \vec{p} \) to \( \vec{q} \) along ray \( r \)
   a. Compute surface normal at \( \vec{p} \)
   b. Apply local shading model

Topic 12:
Basic Ray Tracing

- Introduction to ray tracing
- Computing rays
- Computing intersections
  - ray-triangle
  - ray-polygon
  - ray-quadric
  - the scene signature
- Computing normals
- Evaluating shading model
- Spawning rays
- Incorporating transmission
  - refraction
  - ray-spawning & refraction
Computing the Normal at a Hit Point

Option #1:
- Smoothly interpolate normal from vertices or adjacent faces (e.g., using the linear interpolation technique covered with Phong+ scan conversion)

Option #2:
- For parametric shapes, normal can be evaluated directly at $\mathbf{p}$
  - Implicit form
    \[ \mathbf{n}(\mathbf{p}) = \nabla f(\mathbf{p}) / \| \nabla f(\mathbf{p}) \| \]
  - Explicit form
    \[ \mathbf{n}(\mathbf{p}) = \text{unit vector along } \frac{\partial}{\partial x} S(x, \beta) \times \frac{\partial}{\partial \beta} S(x, \beta) \]

Computing the Normal at a Hit Point

Option #3 (affinely-deformed shapes):
- Let $f(\mathbf{p}) = 0$ be an implicit surface
- Let $M$ be a 4x4 affine transformation matrix
- Suppose we deform the surface by applying $M$ to it
- Point $\mathbf{t}$ will be on the deformed surface if there is a $\mathbf{p}$ such that $\mathbf{t} = M \mathbf{p}$
- \[ \mathbf{p} = M^{-1} \mathbf{t} \]
- Implict eq is
  \[ F(\mathbf{t}) = f(M^{-1} \mathbf{p}) = 0 \]
  \[ \mathbf{n}(\mathbf{t}) = (M^{-1})^T \mathbf{n}(\mathbf{p}) / \| (M^{-1})^T \mathbf{n}(\mathbf{p}) \| \]

See notes for complete proof.
Topic 12:
Basic Ray Tracing

- Introduction to ray tracing
- Computing rays
- Computing intersections
  - ray-triangle
  - ray-polygon
  - ray-quadric
  - the scene signature
- Computing normals
- Evaluating shading model
- Spawning rays
- Incorporating transmission
  - refraction
  - ray-spawning & refraction

Evaluating the Shading Model

Use a two-component model

$$I(\vec{q}) = L(\vec{b}, \vec{n}, \vec{s}) + G(\vec{p}) \cdot r_s$$

- intensity at pixel $\vec{q}$
- local shading model at $\vec{p}$
- global shading component at $\vec{p}$

[Diagram of ray tracing and shading model]
Evaluating the Shading Model

Use a two-component model

\[ I(\vec{q}) = L(\vec{l}, \vec{n}, \vec{s}) + G(\vec{p}) \cdot r_s \]

Phong model

\[ r_n I_n + r_d I_d \max(0, \vec{n} \cdot \vec{s}) + r_s I_s \max(0, \vec{n} \cdot \vec{p}) \]

Computed after ray spawning

Evaluating the Shading Model: Using Textures

Use a two-component model

\[ I(\vec{q}) = L(\vec{l}, \vec{n}, \vec{s}) + G(\vec{p}) \cdot r_s \]

Phong model

\[ r_n I_n + r_d I_d \max(0, \vec{n} \cdot \vec{s}) + r_s I_s \max(0, \vec{n} \cdot \vec{p}) \]

Computed after ray spawning

Texture can be used to modulate \( r_n \) and \( r_d \):
- need to compute \( \vec{p} \)'s texture coordinates
- unlike scan-conversion, we compute \( \vec{p} \)'s texture coordinates by linear interpolation on the polygon plane, not in image space (i.e., no distortion artifacts)
Computational Issues in Basic Ray Tracing

Basic loop:
for each pixel $\bar{q}$
  1. cast ray $r$ through $\bar{q}$
  2. find 1st intersection of $\bar{q}$ with scene (i.e. point $\bar{p}$)
  3. estimate amount of light reaching $\bar{p}$
  4. estimate amount of light travelling from $\bar{p}$ to $\bar{q}$ along ray $r$

a. "spawn" rays $r_1, r_2, ..., r_k$
   from $\bar{p}$ in various directions
b. if ray $r_i$ hits a light source, estimate light travelling along $r_i$ and stop
c. else apply loop recursively to ray $r_i$
Topic 12:

Basic Ray Tracing

- Introduction to ray tracing
- Computing rays
- Computing intersections
  - ray-triangle
  - ray-polygon
  - ray-quadric
  - the scene signature
- Computing normals
- Evaluating shading model
- Spawning rays
- Incorporating transmission
  - refraction
  - ray-spawning & refraction

Whitted Ray Tracing

Basic idea:
- Spawn only one ray
- Ray is along ideal specular direction

1. estimate amount of light reaching $P$
   a. "spawn" rays $r_i$ from $P$ in various specular directions only
   b. if ray $r_i$ hits a light source, estimate light travelling along $r_i$ and stop
   c. else apply loop recursively to ray $r_i$
**Whitted Ray Tracing**

**Motivation:**
- Computationally efficient (1 spawned ray/bounce)
- Models the most important light path (in terms of "light energy" transferred from $r_i$ to $r$)

1. Estimate amount of light reaching $P$
   a. "Spawn" rays $r_i$ from $P$ in various specular directions
   b. If ray $r_i$ hits a light source, estimate light travelling along $r_i$ and stop
   c. Else apply loop recursively to ray $r_i$

---

**Whitted Ray Tracing: An Example**

Use a two-component model

$$I(\mathbf{q}) = L(\mathbf{b}, \mathbf{\hat{n}}, \mathbf{\hat{s}}) + G(\mathbf{p}) \cdot r_s$$

- **Phong model**
  - $r_a I_a + r_d I_d \max(0, \mathbf{\hat{n}} \cdot \mathbf{\hat{d}}) + r_s I_s \max(0, \mathbf{\hat{n}} \cdot \mathbf{\hat{s}})$

- **Global specular term**

---
Whitted Ray Tracing: An Example

Use a two-component model

\[ I(q) = L(\vec{b}, \vec{m}, \vec{s}) + G(\vec{p}) \cdot r_s \]

Phong model

\[ r_a I_e + r_d I_d \ \max(0, \vec{r} \cdot \vec{h}) + r_s I_s \ \max(0, \vec{r} \cdot \vec{s}) \]

Global specular term

\[ L(\vec{b}, \vec{m}, \vec{s}) + G(\vec{p}) r_s \]

Ray \( r_s \) (spawned from \( \vec{p} \))
Whitted Ray Tracing: An Example

Use a two-component model

\[ I(q) = L(b, n, s) + G(p) \cdot r_s \]

**Phong model**

\[ r_n I_n + r_d I_d \max(0, r_s \cdot s) + r_s \max(0, r_s \cdot b) \]

**Global specular term**

\[ L(b, n, s) + G(p) r_s \]

computed by recursively ray tracing along \( r_2 \)

Simulating Shadows

Use a two-component model

\[ I(q) = L(b, n, s) + G(p) \cdot r_s \]

**Phong model**

\[ r_n I_n + r_d I_d \max(0, r_s \cdot s) + r_s \max(0, r_s \cdot b) \]

**Global specular term**

\[ L(b, n, s) \]

if \( p_i \) not visible to light source, include only ambient term

\[ \times \]

computed by recursively ray tracing along \( r_2 \)
Simulating Shadows

Use a two-component model

\[ I(\mathbf{q}) = L(\mathbf{b}, \mathbf{m}, \mathbf{s}) + G(\mathbf{p}) \cdot \mathbf{r}_s \]

**Phong model**

\[ r_a \cdot \mathbf{I}_a + r_d \cdot \mathbf{I}_d \max(0, \mathbf{r}_s) + r_s \cdot \mathbf{I}_s \max(0, \mathbf{r}_s) \]

**Global specular term**

If \( \mathbf{p}_i \) not visible to light source, include only ambient term

\[ \mathbf{r}_a \cdot \mathbf{I}_a \]

computed by recursively ray tracing along \( \mathbf{r}_2 \)

Simulating Shadows

Use a two-component model

\[ I(\mathbf{q}) = L(\mathbf{b}, \mathbf{m}, \mathbf{s}) + G(\mathbf{p}) \cdot \mathbf{r}_s \]

**Phong model**

\[ r_a \cdot \mathbf{I}_a + r_d \cdot \mathbf{I}_d \max(0, \mathbf{r}_s) + r_s \cdot \mathbf{I}_s \max(0, \mathbf{r}_s) \]

**Global specular term**

If \( \mathbf{p}_i \) not visible to light source, include only ambient term

\[ \mathbf{r}_a \cdot \mathbf{I}_a \]

determined by casting a ray from \( \mathbf{p}_i \) to the point light source and testing for object hits between \( \mathbf{p}_i \) and that source
Non-recursive vs. Recursive Ray Tracing

No recursion (local lighting + shadows) 1 Level of recursive spec. reflection

Ray tracing in the movies

2 Levels of recursive spec. reflection 1 Level of recursive spec. reflection

differences occur where reflections of reflections are significant

https://agora.cs.uiuc.edu/download/attachments/10454060/RayTracing_suppl.pdf?version=1
Topic 12:

Basic Ray Tracing

- Introduction to ray tracing
- Computing rays
- Computing intersections
  - ray-triangle
  - ray-polygon
  - ray-quadric
  - the scene signature
- Computing normals
- Evaluating shading model
- Spawning rays
- Incorporating transmission
  - refraction
  - ray-spawning & refraction
Modeling Reflection: Transmission

Transmission:
- Caused by materials that are not perfectly opaque.
- Examples include glass, water and translucent materials such as skin.
Physics of Refraction

\[ \frac{\sin \theta_1}{\sin \theta_2} = \frac{c_1}{c_2} \]

Snell's law

\textbf{Refraction} (bending of rays) occurs when light crosses an interface between two media with different speeds of light.

\textit{Physics:} the speed of light depends on the material through which it travels (and the wavelength of light, but we will ignore that.)
Geometry of Refraction: Transmission Vector

Snell's law
\[ \frac{\sin \theta_1}{\sin \theta_2} = \frac{c_1}{c_2} \]

1. Incident ray, outgoing ray & normal always lie on the same plane \( \Rightarrow \)
d\(\hat{a}\) along \( \frac{c_2}{c_1} \hat{s} + \left[ \frac{c_2}{c_1} \cos \theta_1 - \cos \theta_2 \right] \hat{n} \)

Geometry of Refraction: Path Reversibility

Snell's law
\[ \frac{\sin \theta_1}{\sin \theta_2} = \frac{c_1}{c_2} \]

2. Light paths are always reversible (i.e. light is transmitted exactly the same way if its direction of travel is reversed)
Geometry of Refraction

Snell's law
\[ \frac{\sin \theta_1}{\sin \theta_2} = \frac{c_1}{c_2} \]

3. If \( c_2 < c_1 \), light bends toward the normal.

Geometry of Refraction

Snell's law
\[ \frac{\sin \theta_1}{\sin \theta_2} = \frac{c_1}{c_2} \]

3. If \( c_2 < c_1 \), light bends toward the normal.

If \( c_2 > c_1 \), light bends away from normal.
Geometry of Refraction

Snell's law
\[
\frac{\sin \theta_1}{\sin \theta_2} = \frac{c_1}{c_2}
\]

1. If \( c_2 < c_1 \) light bends toward the normal
2. If \( c_2 > c_1 \) light bends away from normal

Geometry of Refraction: The Critical Angle

Snell's law
\[
\frac{\sin \theta_1}{\sin \theta_2} = \frac{c_1}{c_2}
\]

3. If \( c_2 > c_1 \) there is a critical angle above which no transmission occurs (\( \Rightarrow \) have total internal reflection)
Geometry of Refraction: Total Internal Reflection

Snell's law
\[
\frac{\sin \theta_1}{\sin \theta_2} = \frac{c_1}{c_2}
\]

4. Deriving the critical angle: From Snell's law,
\[
\cos \theta_2 = \sqrt{1 - \left(\frac{c_2}{c_1}\right)^2 \sin^2 \theta_1}
\]
at critical angle, \( \theta_2 = \frac{\pi}{2} \) \( \Rightarrow \) \( \sin \theta_1 = \frac{c_1}{c_2} \)

Geometry of Refraction: Total Internal Reflection

Snell's law
\[
\frac{\sin \theta_1}{\sin \theta_2} = \frac{c_1}{c_2}
\]

4. For \( \theta_1 > \theta_1^* \), \( \theta_2 \) is undefined
Geometry of Refraction: Normal Incidence

Snell's law
\[
\frac{\sin \theta_1}{\sin \theta_2} = \frac{c_1}{c_2}
\]

\( \Theta = 0 \), no bending occurs

\[ \vec{d} \text{ along } -\frac{c_2}{c_1} \vec{z} + \left[ \frac{c_2}{c_1} \cos \theta_1 - \cos \theta_2 \right] \vec{\hat{n}} \]

Topic 12:

Basic Ray Tracing

- Introduction to ray tracing
- Computing rays
- Computing intersections
  - ray-triangle
  - ray-polygon
  - ray-quadric
  - the scene signature
- Computing normals
- Evaluating shading model
- Spawning rays
- Incorporating transmission
  - refraction
  - ray-spawning & refraction
**Whitted Ray Tracing with Refraction**

Basic idea:
- Spawn two rays
  - One ray is along ideal specular direction
  - One is along refraction direction

1. Estimate amount of light reaching P
   a. Spawn rays $r_{i}$ and $s_{i}$ from P in various specular direction and refraction direction
   b. If ray $r_{i}$ hits a light source, estimate light travelling along $r_{i}$ and stop
   c. Else apply loop recursively to ray $r_{i}$

**Whitted Ray Tracing with Refraction**

Much less efficient than specular-only ray tracing because $2^n$ rays are spawned after n bounces (instead of n)
Visualizing the Spawned Rays

Much less efficient than specular-only ray tracing because $2^n$ rays are spawned after $n$ bounces (instead of $n$)

Visualizing ray-spawning as a tree:

Local shading + Reflection + Transmission
Topic 13:

Radiometry

- The big picture
- Measuring light coming from a light source
- Measuring light falling onto a patch: Irradiance
- Measuring light leaving a patch: Radiance
- The Light Transport Cycle
- The Bidirectional Reflectance Distribution Function
The Common Modes of “Light Transport”

The Phong Reflectance Model

Phong model: A simple, computationally-efficient model that has 3 components:
- Diffuse
- Ambient
- Specular
Phong Reflection: The General Equation

\[ L(\vec{b}, \vec{n}, \vec{s}) = r_a I_a + r_d I_d \max(0, \vec{n} \cdot \vec{s}) + r_s I_s \max(0, \vec{n} \cdot \vec{b}) \]

The Common Modes of "Light Transport"

- Specular reflection
- Incident light
- Surface scattering
- Subsurface scattering
- Transmission
- Light source
Generalizing the Phong Model

- All reflected light can be thought of as a form of scattering.
- For most real materials, the Phong-based distinction into specular and diffuse reflection is a crude approximation.

Generalizing the Phong Model: How?

- Seek to answer the following question:
  given a specific incident direction, how much light is reflected along a specific outgoing direction?
Generalizing the Phong Model: How?

. Seek to answer the following question:
given a specific incident direction
how much light is reflected along a specific outgoing direction?

The BRDF of a Surface Point (in 2D)

BRDF = Bidirectional Reflectance Distribution Function

- It is a function $\rho_F : \left[ \frac{\pi}{2}, \frac{\pi}{2} \right] \times \left[ \frac{\pi}{2}, \frac{\pi}{2} \right] \rightarrow [0,1]$ 

$$\rho_F(\theta_r, \theta_o) = \frac{\text{emitted light in direction } \theta_o}{\text{incident light in direction } \theta_r}$$

incoming direction outgoing direction
The BRDF of a Surface Point (in 3D)

\[ \rho : [-n, n] \times [0, \pi / 2] \times [-n, n] \times [0, \pi / 2] \rightarrow [0,1] \]

\[ \rho (\vec{d}_i, \vec{d}_o) = \frac{\text{emitted light in direction } \vec{d}_o}{\text{incident light in direction } \vec{d}_i} \]

incoming direction outgoing direction
Measuring BRDFs with a Gonioreflectometer

Photo: M3L New Zealand

Visualizing BRDFs

The MERL BRDF database